LPCXpresso 1769 CPU Module SPI Flash Memory Testing

Troy Kurniawan
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 95192
E-mail: troy.kurniawan@sjsu.edu

Abstract

This report discusses a lab that creates a system using an LPCXpresso 1769 CPU module to realize input power regulation, GPIO input and output, and SPI flash memory testing/debugging function. The main challenges were to understand the CPU module and sub-circuits' physical capabilities and how to write code to use the GPIO pins and SPI interface. The lab was successful in creating a power regulating circuit along with GPIO input/output pins and an SPI interface.

1. Introduction

In this lab, a prototype microprocessor board was designed using the LPCXpresso 1769 CPU [1]. This board is a CPU module that has many capabilities that can be programed through a computer using a software IDE called "MCUXpresso." For this lab, different objectives were called for in the form of various mini circuits. The board contains a GPIO input and output testing/debugging function and a power regulator to supply power. In addition, the board contains an AT45 Flash memory which is able to store data obtained through an SPI interface [2]. Furthermore, C code was written to program the board and its functionality. Much of the theory work was provided by the professor, which helped guide this lab towards its success [3]. However, other challenges were also present, such as building the physical border using a soldering kit. Plenty of help from outside resources were available to assist in the lab.

One way to power the board is through the power regulator circuit. An input signal can be applied to the power regulator circuit and the output of the circuit is 5V. This 5V power is applied to the CPU module.

To test the functionality of the CPU board's GPIO pins, two circuits were constructed. The GPIO input testing circuit essentially contains a switch that is able to apply a 5V power to one of the GPIO pins on the CPU module. The software is then able to detect whether or not this signal is applied and notifies the user in the IDE console. The GPIO output testing circuit contains a simple LED circuit that can be powered by the output pin of the CPU module. The user is able to control the output using the IDE console. When the output is active, the LED lights up. When the output is not active, the LED does not light up.

The AT45 Flash memory was added to the board to give the system memory capabilities. For this lab, the user was able to input a specific text message into the IDE console, which was then stored into the Flash memory. The user was then able to read back from the Flash memory.

This report is split into a variety of sections to cover the project's methodology, hardware and software implementation, and conclusion.

2. Methodology

This methodology section is split into two sections which include the "Objectives and Technical Challenges" section and the "Problem Formulation and Design" section. The first section presets the various goals the lab aims to achieve, and the second section tells of the approach to designing the board and code.

2.1. Objectives and Technical Challenges

The main task for this lab was to design a functional prototype board with a CPU module. various other objectives were also presented. The first was to create a power regulating circuit. This circuit would power the board without the need for the cable that connects directly to the CPU module.

The second objective was to create a GPIO input testing circuit. This circuit would be used to test whether the CPU module can detect incoming voltage signals.

The third objective was to create a GPIO output testing circuit. This circuit would be used to test whether the CPU module can produce a voltage output to turn on an LED.

The fourth task was to successfully interface the SPI Flash memory with the CPU module. It would be used to store a simple text message written by the user in the code. When the CPU module is turned off, the Flash memory would retain the information written to it.

The fifth task was to write code using the MCUXpresso IDE so that the CPU module can control the GPIO pins and store data into the Flash memory.

Various other technical tasks are also presented. Soldering and wire wrapping are two prevalent challenges that appeared for this lab.

2.2. Problem Formulation and Design

The board's design was based on a design provided by the professor. Although this is true, much of the layout was up to the builder of the board. The challenge was to lay out the sections of the circuit in a way that optimized

space and used up the least resources. For this design, the CPU module was placed in the middle so that different circuits can be placed on both sides to access the pins on both sides. All other circuits were placed in their respective spots so that they could access their appropriate pins quickly. The full layout can be found in figures 1 and 2.

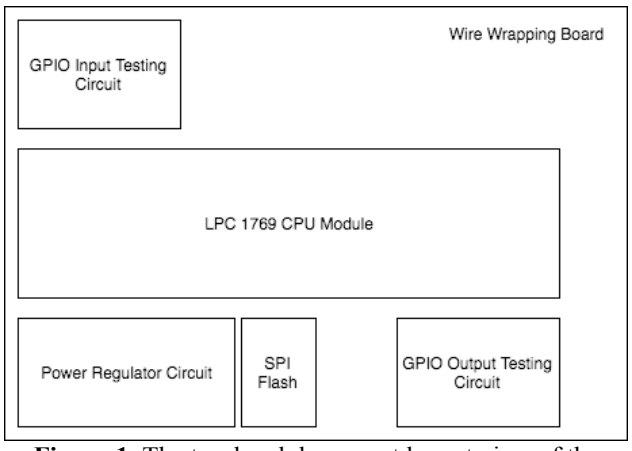

Figure 1. The top-level drawn out layout view of the prototype board.

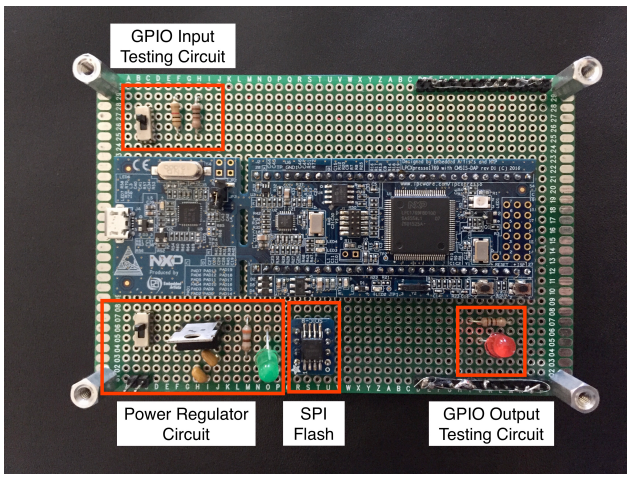

Figure 2. The top-level physical layout view of the prototype board.

The power regulator circuit is located at the bottom left of the board. It was placed here in order to reach the necessary pins of the CPU module (pins one and two) easily. Its entire circuit contains a power connector, an SPDT switch, and LM7805 power regulator, and an LED indicator circuit. It connects to the J2-2 pin of the CPU module. This layout can be found in figure 3.

The GPIO input testing circuit is located at the top left of the board. It connects to the J2-21 pin of the CPU module. Its connections can be found in figure 3.

The GPIO output testing circuit is located at the bottom right of the board. It connects to the J2-22 pin of the CPU module. Its connections can be found in figure 3.

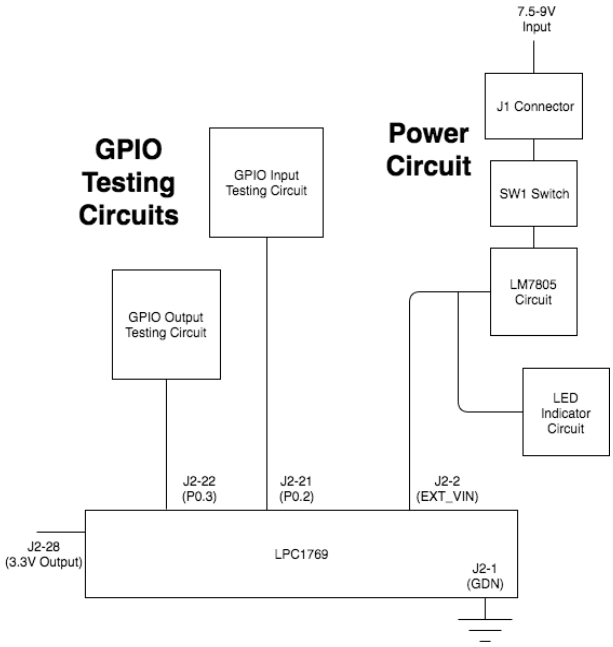

Figure 3. The system block diagram of the GPIO testing circuits and the power circuit connected to the LPC1769 module.

Different special purpose registers (SPRs) were used for GPIO control. The SPR "FIODIR" sets the appropriate pin as an output or input controlled pin. The SPR "FIOSET" sets the output of the GPIO port to be a high voltage value. The SPR "FIOCLR" sets the output of the GPIO port to be a low voltage value. The SPR "FIOPIN" reads the voltage value from the GPIO input and determines whether it should store a true value "1" or a false value "0".

The AT45 Flash memory is located at the bottom of the board. Its various pins connect to various pins on the CPU module. The "SI" pin of the memory stands for "serial input" and acts as the data input for the memory. It is connected to pin J2-5 of the CPU module, which is the "MOSI1" pin (master output, slave input). The "SCK" pin of the memory is the clock input. It is connected to the J2-7 pin of the CPU module, which is the "SCK1" pin. The "RESET" pin of the memory is the reset input used to wipe the memory. It is left floating and unused. The "CS" pin of the memory is the "chip select" input. It is connected to the J2-7 pin of the CPU module, which is the "SSEL1" pin. The "WP" pin of the memory is the "write protect" input. It is connected to the J2-28 pin of the CPU module, which is the 3.3V power supply. The "VCC" pin of the memory is the power input. It is also connected to the J2-28 pin of the CPU module, which is the 3.3V power supply. The "GND" pin of the memory is the

ground. It is connected to the J2-1 pin of the CPU module, which is the ground. The "SO" pin of the memory is the "serial output" pin, which acts as the data output. It is connected to the J2-6 pin of the CPU module, which is the "MISO1" pin (master input, slave output). Its connections can be found in figure 4.

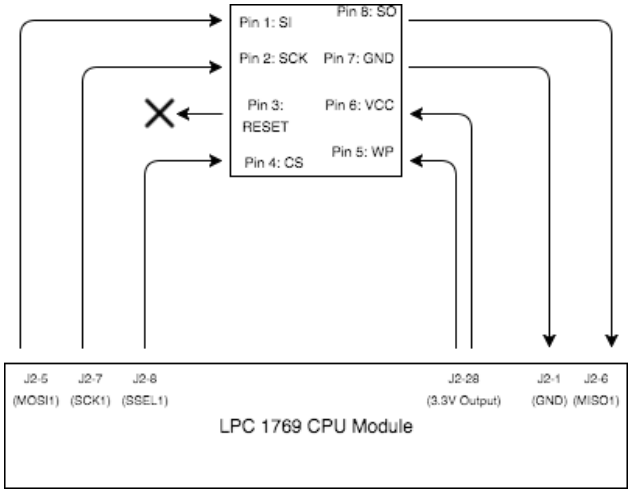

Figure 4. The system block diagram of the AT45 Flash memory connected to the LPC1769 module.

Further details about the various circuits can be found in the implementation section of this report.

The code for the GPIO pins was fairly simple. A while loop was used so that the code can be reused infinitely. The main loop contains two parts: one that covers the input circuit and one that covers the output circuit. The input circuit section simply contains an if and else statement that checks whether or not a voltage is received. The output circuit section also contains if and else statements, but the requirements for those statements are based on user input. More details about this code can be found in the implementation section of this report. Pseudocode can be found in the appendix section of this report as well.

The code for the SPI Flash memory was designed to write the message "SJSU CMPE127 <Name> <Student ID>" to the memory. A while loop was used so that the code can be reused infinitely. The main loop contains two parts: one that covers writing from the CPU module to buffer 1 and then the memory and one that writes from the memory to buffer 2 and reads from the memory to the CPU module. The user can select "1" to write to memory or "2" to read from memory. More details about this code can be found in the implementation section of this report. Pseudocode can be found in the appendix section of this report as well.

3. Implementation

This implementation section is organized into two sections which include the hardware's design and the software's design. The hardware's design was based off of a schematic provided by the professor. Additions and edits were made to that original template. The software's design was also based on previous work done by the professor. Again, changes were made to the original work to reflect the intended design.

3.1. Hardware Design

For the power circuit, a USB cable was used to supply the source power to the rest of the circuit. The cable is connected to an SPDT switch to control the circuit. The switch is also connected to pin one of the LM7805 power regulator and a 0.33µF capacitor. Pin two of the power regulator is also connected to the 0.33µF capacitor, the 0.1µF capacitor, and ground. Pin three of the power regulator is connected to the 0.1µF capacitor and port "J2-2" of the LPC1769 module to successfully power it. Pin three is also connected to a simple LED circuit to indicate its successful functionality. This LED circuit consists of a 390Ω resistor a green LED, and ground all connected in series. The circuit can be found in figure 5.

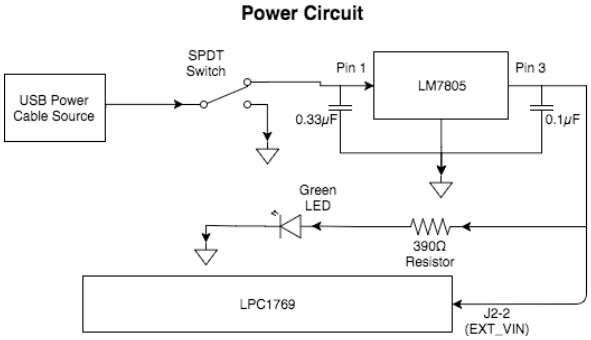

Figure 5. The schematic of the power circuit connected to the LPC1769 module.

For the input testing circuit, port "J2-28" of the LPC1769 module was used to supply a 3.3V power to an external circuit. The circuit consists of a SPDT switch to control its operation. The switch is then connected to a 1kΩ resistor and a 100Ω resistor in series. While the 100Ω resistor connects to ground, the the 1kΩ resistor also connects to pin 0.2 of the module. The circuit can be found in figure 6.

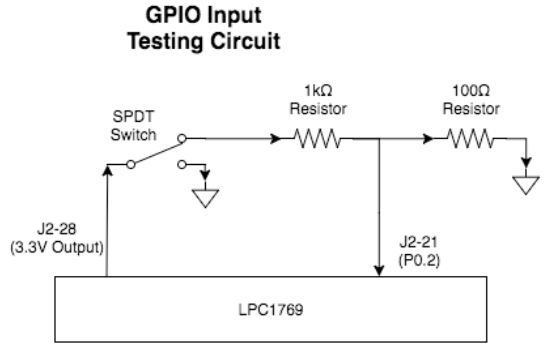

Figure 6. The schematic of the GPIO input testing circuit connected to the LPC1769 module.

For the output testing circuit, pin 0.3 was used to output power to a simple LED circuit. The circuit consisted of a 100Ω resistor, a red LED, and ground all connected in series. The circuit can be found in figure 7.

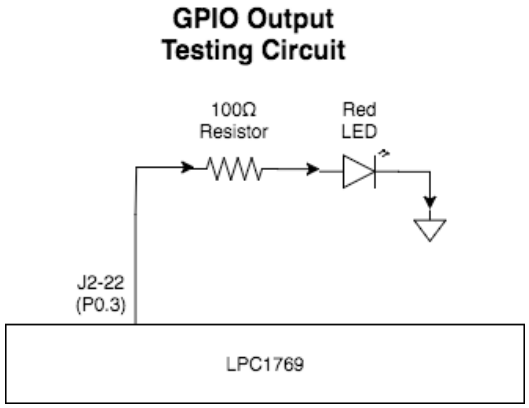

Figure 7. The schematic of the GPIO output testing circuit connected to the LPC1769 module.

The AT45 Flash memory schematic is simply a direct connection between the chip's pins and the CPU module's pins. The same connections can be found in figure 4.

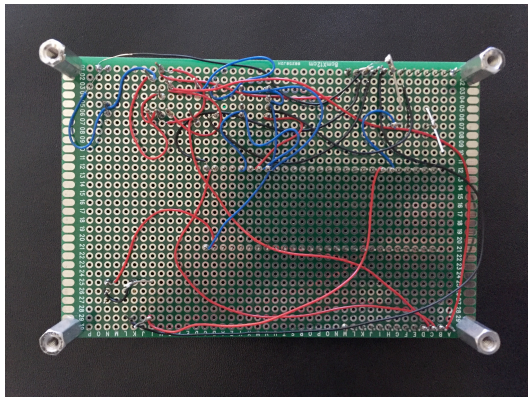

Figure 8. The under side of the board, showing the various connections.

Bill of Material (List of Components):

- PCB Board
- 1/2in. Standoffs
- LPC1769 Xpresso CPU Module
- 30AWG Wires
- LM7508 Voltage Regulator
- Green LED
- Red LED
- 0.1µF Capacitor
- 0.33µF Capacitor
- Two SPDT Switches
- One 390Ω Resistor
- Two 100Ω Resistor
- One 1kΩ Resistor
- AT45 Flash Memory
- Header Pins
- USB Cable
- Solder

The full kit containing all the parts totaled to $75.

Table 1. Connection Table that lists the different pins on the CPU module and what it's connected to on the board.

LPC1769 Pin:	**Connection:**
J2-1 (Ground)	Power Regulator Circuit GPIO Input Circuit GPIO Output Circuit AT45's GND
J2-2 (VCC)	Power Regulator Circuit
J2-5 (MOSI1)	AT45's SI
J2-6 (MISO1)	AT45's SO
J2-7 (SCK1)	AT45's SCK
J2-8 (SSEL1)	AT45's CS
J2-21 (GPIO P0.2)	GPIO Input Circuit
J2-22 (GPIO P0.3)	GPIO Output Circuit
J2-28 (3.3V Output)	GPIO Input Circuit AT45's WP and VCC

3.2. Software Design

For the GPIO pins, the code begins by setting pin 0.2 as the input and pin 0.3 as the output. From there, the code enters an infinite while loop. The loop first attempts to detect a voltage signal at pin 0.2. If a signal is detected, the console in MCUXpresso will print a statement saying "Input received." If a signal is not detected, the console will print a statement saying "No input found." The loop then asks the user to input either a "1" or a "0" into the console. If the user chooses "1," the LED in the GPIO output testing circuit (pin 0.3) will turn on. If the user chooses "0," the LED in the GPIO output testing circuit will turn off. The loop then repeats. A flow chart can be found in figure 9.

Algorithmic Procedure:

1. Set up the GPIO input and output pins
2. Detect whether or not an input signal is received from the GPIO input.
3. Print to the console accordingly.
4. Prompt user to either turn on the LED or turn off the LED.
5. Set the LED's status accordingly.

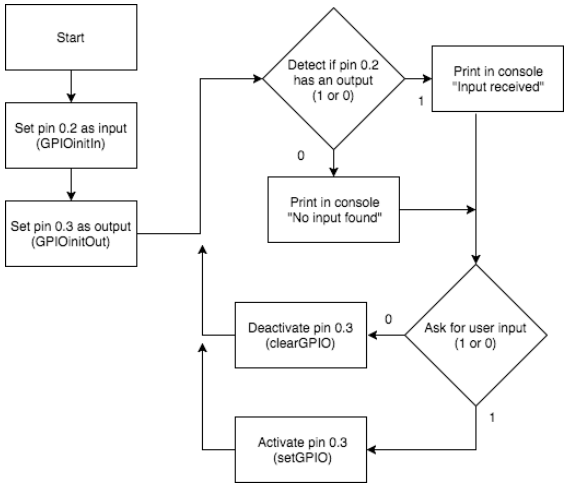

Figure 9. The C code main function flowchart for the GPIO input and output testing/debugging.

GPIO Pseudocode:

```
while(1){
    Ask for user input

    if(Voltage detected at pin 0.2){
        Input received!
    }
    else{
        No input received.
    }

    if(User chooses 1){
        Turn on LED
    }
    else if(User chooses 0){
        Turn off LED
    }
}
```

For the Flash memory, the code begins by setting up the various pins such as MISO, MOSI, SCK, and SSEL. From there, the code enters an infinite while loop. The user is then prompted to enter a value of either "1" or "2". From there, two if statements are setup. If the user entered a "1", the code would perform a function that would write the text message from the code to buffer 1, read from buffer 1, then write to the memory. If the user entered a "2", the code would perform a function that would write the text message from the memory to buffer 2 and read from buffer 2. The loop then repeats. A flow chart can be found in figure 10.

Algorithmic Procedure:

1. Prompt the user to either write from the code to the memory and read from buffer 1 or read from the memory.
2. If the user chooses to write, the code would write the text message to buffer 1, read from buffer 1, then write to memory.
3. If the user chooses to read, the code would write the text message from the memory to buffer 2, and then read from buffer 2.
4. Repeat

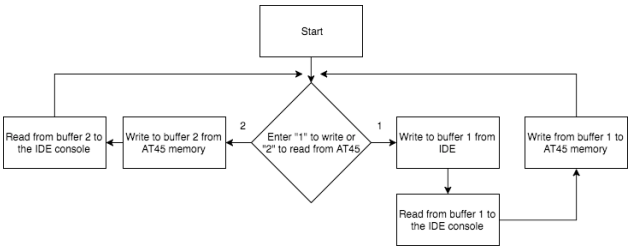

Figure 10. The C code main function flowchart for the AT45 Flash memory.

SPI Flash Memory Pseudocode:

Set up MISO, MOSI, SCK, and SSEL pins.

```
while(1){
    Ask for user input

    if(User chooses 1){
        Write to buffer 1 from console
          Read from buffer 1 to console
          Write to Flash memory from buffer 1
    }
    else if(User chooses 0){
        Write to buffer 2 from memory
          Read from buffer 2 to console
    }
}
```

Source code for all circuits can be found in the Appendix section of this report.

4. Testing and Verification

The prototype board was tested and verified in various ways. To test the power circuit's functionality, a green LED was placed at the end of the circuit to signify its status. When the system was powered, the LED would turn on. This green LED can be seen in figure 11.

To test the board's GPIO input circuit, a code was written in MCUXpresso that sensed whether or not a signal was received from its specified pin. An SPDT switch was used for the input circuit to control whether a signal was applied to this pin. When the switch was off, the IDE printed a statement saying a signal was not received. When the switch was on, the IDE printed a statement saying a signal was received. Verification can be found in figures 11 to 13.

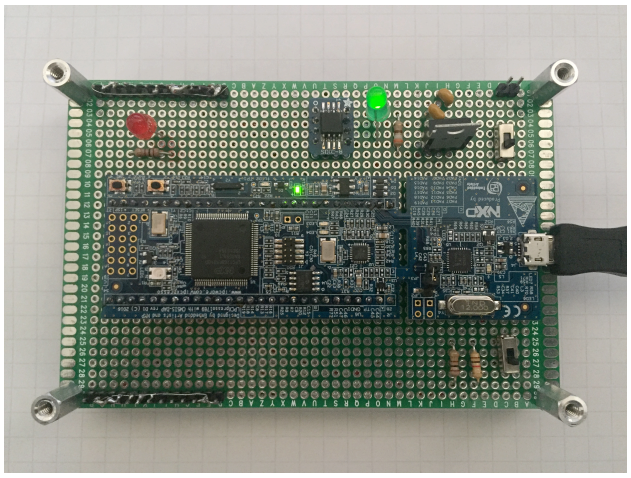

Figure 11. The prototype board with the green LED power indicator turned on, verifying its powered status. The GPIO input circuit on the bottom right is also switched off, applying no input signal.

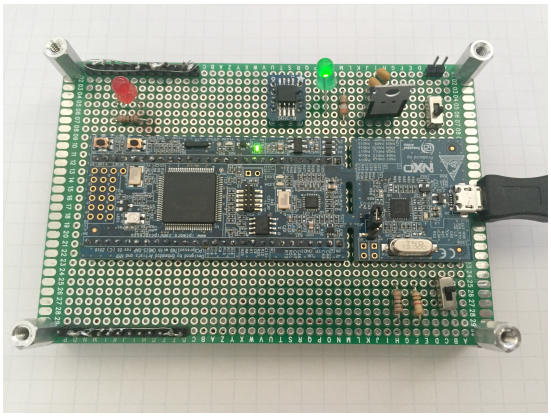

Figure 12. The prototype board with the GPIO input circuit on the bottom right switched on, applying an input signal.

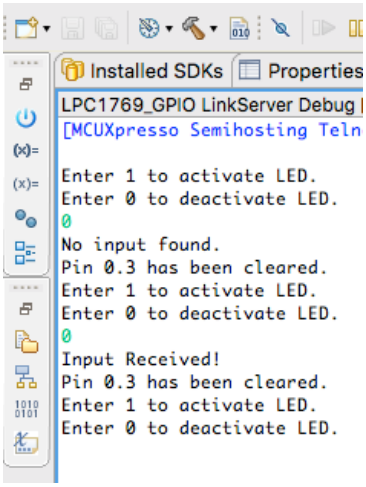

Figure 13. The MCUXpresso IDE console that shows that when the GPIO input switch is off, the console reads "No input found." and when the switch is on, the console reads "Input Received!"

To test the board's GPIO output circuit, a code was written that could control whether a pin could output a signal. When the user typed a "1" in the console, the output pin would turn on. When the user typed a "0" in the console, the output pin would turn off. A red LED was placed at the end of the circuit to signify its status. Verification can be found in figures 14 and 15.

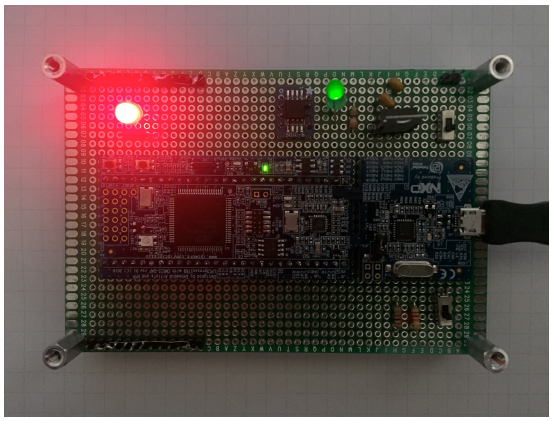

Figure 14. The prototype board with the GPIO output circuit on the top left switched on. The red LED is active.

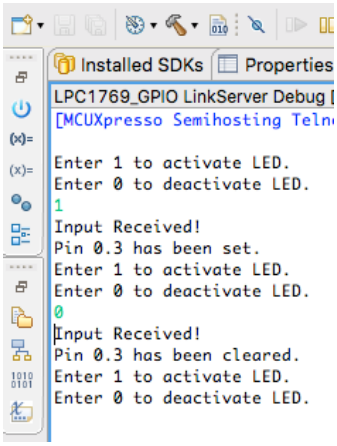

Figure 15. The MCUXpresso IDE console that shows that when the user inputs a "1", the red LED is active and the console prints out "Pin 0.3 has been set." When the user inputs a "0", the red LED is not active and the console prints out "Pin 0.3 has been cleared."

To test the board's Flash memory circuit, a code was written in MCUXpresso that wrote to the memory and read from the memory. After writing to the memory, the board was unplugged, turning the system off. The board was then plugged in again. Once it booted up, the read function was started to see if the Flash memory retained the information. Verification can be found in figures 16 and 17.

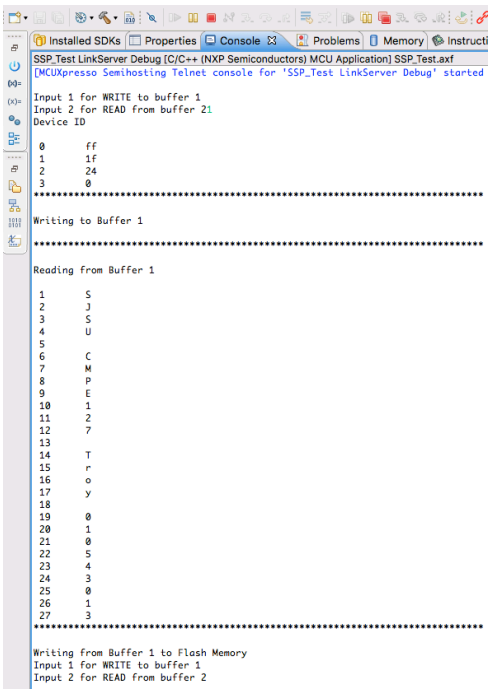

Figure 16. The MCUXpresso IDE console that shows that when the user inputs a "1", the IDE writes the text message to buffer 1, reads from buffer 1, and then writes from buffer 1 to the memory.

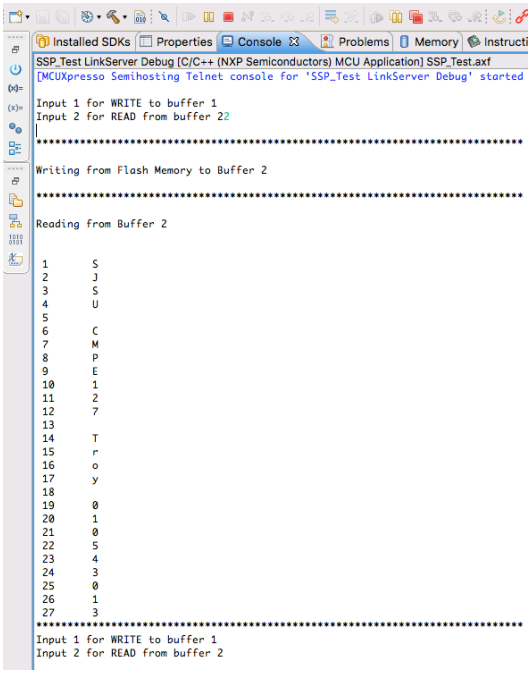

Figure 17. The MCUXpresso IDE console that shows that when the user inputs a "2", the IDE writes the text message from memory to buffer 2 and reads from buffer 2. Notice how the code did not write to memory first.

5. Conclusion

The lab began with a simple task to design a prototype microprocessor board with a power regulating circuit, a GPIO input/output testing/debugging circuit, and a SPI Flash memory circuit. The objectives laid out for this lab were all met successfully. The power regulator circuit powers the board. The GPIO input circuit is able to apply a signal to the board, and the board is able to detect it. The GPIO output circuit is able to apply power to an outside circuit (in this case, an LED circuit). The SPI Flash memory circuit is able to store a text message within the memory and retain it despite the powered status of the CPU module. The lab was a great introduction to microprocessor design, specifically when it comes to designing different circuit to support the board. The process of manually putting the board and circuits together was albeit lengthy and challenging, but it was solid practice for future projects and labs. Overall, the lab was a great success.

6. Acknowledgement

Thank you San Jose State University's Software & Engineering Society (SCE) for providing the necessary tools and equipment (solder, soldering iron, parts kit, etc) to complete this prototype board.

7. References

[1] LPCXpresso 1769 CPU Module Datasheet
[2] AT45 Flash Memory Datasheet
[3] H. Li, GitHub Files

8. Appendix

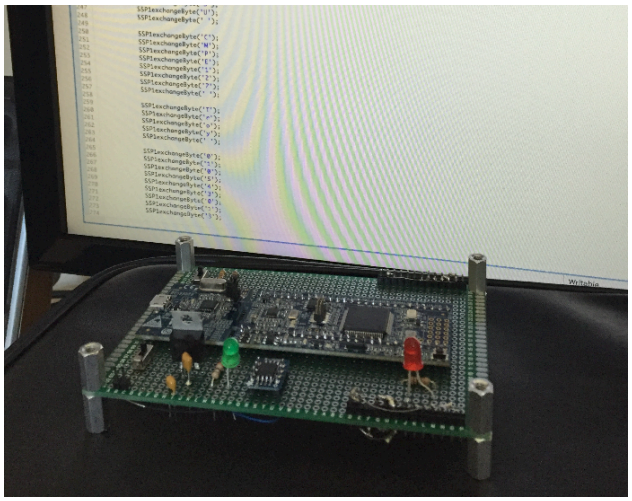

Figure 18. The prototype board next to the MCUXpresso Code

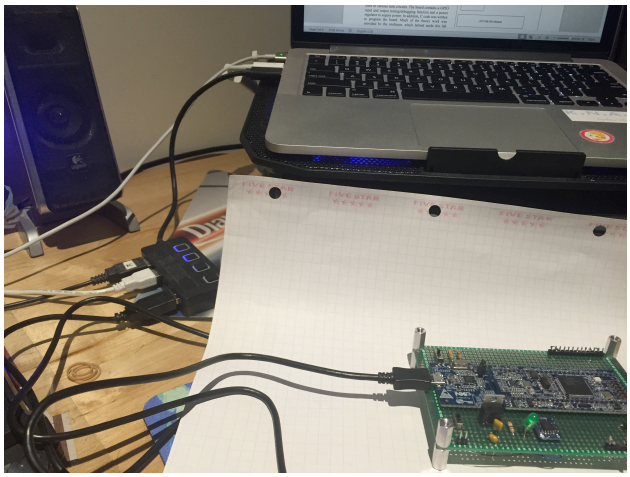

Figure 19. The prototype board connected to the computer

GPIO Input and Output Circuit's Main Source Code:

```
#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include <cr_section_macros.h>
#include <stdio.h>

//Initialize the port and pin as outputs.
void GPIOinitOut(uint8_t portNum, uint32_t pinNum)
{
		if (portNum == 0)
		{
			LPC_GPIO0->FIODIR |= (1 << pinNum);
		}
		else if (portNum == 1)
		{
			LPC_GPIO1->FIODIR |= (1 << pinNum);
		}
		else if (portNum == 2)
		{
			LPC_GPIO2->FIODIR |= (1 << pinNum);
		}
		else
		{
			puts("Not a valid port!\n");
		}
}

//Initialize the port and pin as inputs.
void GPIOinitIn(uint8_t portNum, uint32_t pinNum)
{
		if (portNum == 0)
		{
			LPC_GPIO0->FIODIR &= ~(1 << pinNum);
		}
		else
		{
			puts("Not a valid port!\n");
		}
}

void setGPIO(uint8_t portNum, uint32_t pinNum)
{
		if (portNum == 0)
		{
			LPC_GPIO0->FIOSET = (1 << pinNum);
			printf("Pin 0.%d has been set.\n",pinNum);
		}
		//Can be used to set pins on other ports for future
modification
		else
		{
			puts("Only port 0 is used, try again!\n");
		}
}

//Deactivate the pin
void clearGPIO(uint8_t portNum, uint32_t pinNum)
{
		if (portNum == 0)
		{
			LPC_GPIO0->FIOCLR = (1 << pinNum);
			printf("Pin  0.%d  has  been  cleared.\n",
pinNum);
		}
		//Can be used to clear pins on other ports for future
modification
		else
		{
			puts("Only port 0 is used, try again!\n");
		}
}

int main(void)
{
    // Force the counter to be placed into memory
    volatile static int i = 0 ;
    //Set pin 0.2 as input
		GPIOinitIn(0,2);
		//Set pin 0.3 as output
		GPIOinitOut(0,3);
		//Sense input
		int input = 0;

    while(1)
    {
		printf("Enter  1  to  activate  LED.\nEnter  0  to
deactivate LED.\n");
	    scanf("%d", &i);
```

```
        //Detect pin 0.2
        input = LPC_GPIO0->FIOPIN & (1 << 2);

        //================================

          if(input)
          {
                    //Senses power from pin 0.2
                    printf("Input Received!\n");
          }
          else
          {
                    //Loses power from pin 0.2
                    printf("No input found.\n");
          }

          //================================

                    if (i == 1)
                    {
                              //Activate pin 0.3
                              setGPIO(0,3);
                    }
                    else if (i == 0){
                              //Clear pin 0.3
                              clearGPIO(0, 3);
                    }
                    else
                    {
                              puts("Not a valid option!\n");
                    }
    }
    //0 should never be returned, due to infinite while loop
    return 0;
}
```

AT45 Flash Memory Circuit's Main Source Code:

```
/*
===============================================================
==============
 Name        : SSP_Test.c
 Author      : $(author)
 Version     :
 Copyright   : $(copyright)
 Description : main definition
===============================================================
==============
*/

#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include <cr_section_macros.h>

#include <stdio.h>
#include "ssp.h"
#include <stdio.h>

/* Be careful with the port number and location number, because
some of the location may not exist in that port. */
#define PORT_NUM                        1
#define LOCATION_NUM                    0
#define USE_CS                                  1
#define USE_NCS                                 0
#define OP_ID               0x9F
uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];

/*************************************************************
*************
** Function name:             LoopbackTest
**
** Descriptions:              Loopback test
**
** parameters:                        None
** Returned value:            None
**
*************************************************************
************/
void LoopbackTest( uint32_t portnum, uint32_t location )
{
  uint32_t i;

#if !USE_CS
  /* Set SSEL pin to output low. */
  SSP_SSELToggle( portnum, 0 );
#endif
  i = 0;
  while ( i <= SSP_BUFSIZE )
  {
          /* to check the RXIM and TXIM interrupt, I send a
block data at one time
          based on the FIFOSIZE(8). */
          SSPSend( portnum, (uint8_t *)&src_addr[i], FIFOSIZE );
          /* If RX interrupt is enabled, below receive routine
can be
          also handled inside the ISR. */
          SSPReceive(    portnum,    (uint8_t    *)&dest_addr[i],
FIFOSIZE );
          i += FIFOSIZE;
  }
#if !USE_CS
  /* Set SSEL pin to output high. */
  SSP_SSELToggle( portnum, 1 );
#endif

  /* verifying write and read data buffer. */
  for ( i = 0; i < SSP_BUFSIZE; i++ )
  {
          if ( src_addr[i] != dest_addr[i] )
          {
            while( 1 );                             /*
Verification failed */
          }
  }
  return;
}

/*************************************************************
*************
** Function name:             SEEPROMTest
**
** Descriptions:              Serial EEPROM(Atmel 25xxx) test
**
** parameters:                        None
** Returned value:            None
**
*************************************************************
************/
void SEEPROMTest( uint32_t portnum, uint32_t location )
{
  uint32_t i, timeout;
#if SSP_DEBUG
  uint8_t temp[2];
#endif

  if ( portnum == 1 )
  {
          LPC_GPIO0->FIODIR |= (0x1<<16);                   /*
SSP1, P0.16 defined as Outputs */
  }
  else
  {
          LPC_GPIO0->FIODIR |= (0x1<<6);                    /*
SSP0 P0.6 defined as Outputs */
  }
  SSP_SSELToggle( portnum, 0 );

  /* Test Atmel 25016 SPI SEEPROM. */
  src_addr[0] = WREN;                               /*   set   write
enable latch */
  SSPSend( portnum, (uint8_t *)src_addr, 1 );
  SSP_SSELToggle( portnum, 1 );

  for ( i = 0; i < DELAY_COUNT; i++ );  /*  delay  minimum  250ns
*/
```

```
  SSP_SSELToggle( portnum, 0 );
  src_addr[0] = RDSR;         /* check status to see if write
enabled is latched */
  SSPSend( portnum, (uint8_t *)src_addr, 1 );
  SSPReceive( portnum, (uint8_t *)dest_addr, 1 );
  SSP_SSELToggle( portnum, 1 );
  if ( dest_addr[0] & (RDSR_WEN|RDSR_RDY) != RDSR_WEN )
  /* bit 0 to 0 is ready, bit 1 to 1 is write enable */
  {
          while ( 1 );
  }

  for ( i = 0; i < SSP_BUFSIZE; i++ )   /* Init RD and WR buffer
*/
  {
          src_addr[i+3] = i;  /* leave three bytes for cmd and
offset(16 bits) */
          dest_addr[i] = 0;
  }

  /* please note the first two bytes of WR and RD buffer is used
for
  commands and offset, so only 2 through SSP_BUFSIZE is used for
data read,
  write, and comparison. */
  SSP_SSELToggle( portnum, 0 );
  src_addr[0] = WRITE;        /* Write command is 0x02, low 256
bytes only */
  src_addr[1] = 0x00;         /*  write  address  offset  MSB  is
0x00 */
  src_addr[2] = 0x00;         /*  write  address  offset  LSB  is
0x00 */
  SSPSend( portnum, (uint8_t *)src_addr, SSP_BUFSIZE );
  SSP_SSELToggle( portnum, 1 );

  for ( i = 0; i < 0x30000; i++ );      /* delay, minimum 3ms */

  timeout = 0;
  while ( timeout < MAX_TIMEOUT )
  {
          SSP_SSELToggle( portnum, 0 );
          src_addr[0] = RDSR; /*  check  status  to  see  if  write
cycle is done or not */
          SSPSend( portnum, (uint8_t *)src_addr, 1);
          SSPReceive( portnum, (uint8_t *)dest_addr, 1 );
          SSP_SSELToggle( portnum, 1 );
          if ( (dest_addr[0] & RDSR_RDY) == 0x00 )          /*
bit 0 to 0 is ready */
          {
              break;
          }
          timeout++;
  }
  if ( timeout == MAX_TIMEOUT )
  {
          while ( 1 );
  }

  for ( i = 0; i < DELAY_COUNT; i++ );  /*  delay,  minimum  250ns
*/

  SSP_SSELToggle( portnum, 0 );
  src_addr[0] = READ;                   /* Read command is 0x03,
low 256 bytes only */
  src_addr[1] = 0x00;                   /*  Read  address  offset
MSB is 0x00 */
  src_addr[2] = 0x00;                   /*  Read  address  offset
LSB is 0x00 */
  SSPSend( portnum, (uint8_t *)src_addr, 3 );
  SSPReceive(  portnum,  (uint8_t  *)&dest_addr[3],  SSP_BUFSIZE-3
);
  SSP_SSELToggle( portnum, 1 );

  /* verifying, ignore the difference in the first two bytes */
  for ( i = 3; i < SSP_BUFSIZE; i++ )
  {
          if ( src_addr[i] != dest_addr[i] )
          {
            while( 1 );                         /*
Verification failed */
          }
  }
  return;
}


void initSSP1(void){
    //power up spi1
    LPC_SC->PCONP |= (1<<10);
    //01 PCLK_peripheral = CCLK.. 01, since it's CCLK/1
    LPC_SC->PCLKSEL0 &= ~(3<<20);
    LPC_SC->PCLKSEL0 |= (1<<20);

    //P0.6 is used as a GPIO output and acts as a Slave select
    LPC_GPIO0->FIODIR |= (1<<6);
    LPC_GPIO0->FIOSET = (1<<6);
    //P0.7:9 init
    LPC_PINCON->PINSEL0 &= ~((3<<18)|(3<<16)|(3<<14));
    LPC_PINCON->PINSEL0 |= ((2<<18)|(2<<16)|(2<<14));
    //data size set to 8 bits
    LPC_SSP1->CR0 = 0x07;
    //For AT45 flash SI pin is always latched on the rising edge
of SCK, while output data
    //on the SO pin is always clocked out on the falling edge of
SCK.
    //MS=0 (Master), SSE =1
    LPC_SSP1->CR1 = 0x2;
    LPC_SSP1->CPSR = 8; //SCK Frequency for Continuous Array
Read(Low Frequency) is 33Mhx max. here we are setting it below
it.
}

uint8_t SSP1exchangeByte(uint8_t out){
    LPC_SSP1->DR = out;
    while(LPC_SSP1->SR & (1<<4));
    return LPC_SSP1->DR;
}
/*****************************************************************
***************
**   Main Function
*****************************************************************
*************/
#define SPI_DELAY 1000
int main (void)
{
          while(1){

          int user;

          printf("Input 1 for WRITE to buffer 1\nInput 2 for
READ from buffer 2");
          scanf("%i", &user);

          if(user==1){

          int i;
          initSSP1();
                  //enable_timer(0);
                  for ( i = 0; i < 10000; i++ );
                  printf("Device ID\n");
                  LPC_GPIO0->FIOCLR = (1<<6);
                  printf("\n 0\t %x",SSP1exchangeByte(0x9f));
                  printf("\n 1\t %x",SSP1exchangeByte(0x9f));
                  printf("\n 2\t %x",SSP1exchangeByte(0x00));
                  printf("\n 3\t %x",SSP1exchangeByte(0x00));
                  LPC_GPIO2->FIOSET = (1<<6);
                  for ( i = 0; i < 10000; i++ );

printf("\n*****************************************************
**********************\n");

                  //*************************Write to buffer1
                  //Opcode for write to buff1
                  initSSP1();
                  printf("\nWriting to Buffer 1\n");
                  LPC_GPIO0->FIOCLR = (1<<6);
                  SSP1exchangeByte(0x84); // Opcode Write
                  SSP1exchangeByte(0x00); //
                  SSP1exchangeByte(0x00);
                  SSP1exchangeByte(0x00);
```

```
                    //SSP1exchangeByte('0');
                    SSP1exchangeByte('S');
                    SSP1exchangeByte('J');
                    SSP1exchangeByte('S');
                    SSP1exchangeByte('U');
                    SSP1exchangeByte(' ');

                    SSP1exchangeByte('C');
                    SSP1exchangeByte('M');
                    SSP1exchangeByte('P');
                    SSP1exchangeByte('E');
                    SSP1exchangeByte('1');
                    SSP1exchangeByte('2');
                    SSP1exchangeByte('7');
                    SSP1exchangeByte(' ');

                    SSP1exchangeByte('T');
                    SSP1exchangeByte('r');
                    SSP1exchangeByte('o');
                    SSP1exchangeByte('y');
                    SSP1exchangeByte(' ');

                    SSP1exchangeByte('0');
                    SSP1exchangeByte('1');
                    SSP1exchangeByte('0');
                    SSP1exchangeByte('5');
                    SSP1exchangeByte('4');
                    SSP1exchangeByte('3');
                    SSP1exchangeByte('0');
                    SSP1exchangeByte('1');
                    SSP1exchangeByte('3');

                    LPC_GPIO2->FIOSET = (1<<6);
                    for ( i = 0; i < 10000; i++ );

printf("\n**************************************************
**********************\n");

                        initSSP1();
                        printf("\nReading from Buffer 1\n");
                                    LPC_GPIO0->FIOCLR        =
(1<<6);
                      //*************************Read from buff
1
                                                //Opcode    for
Read buff 1

                    SSP1exchangeByte(0xd4);

                    //Send 3 Bytes = 24 bits= 16 don't care bits
+ 8 buffer addr bits ????

                    SSP1exchangeByte(0x00);

                    SSP1exchangeByte(0x00);

                    SSP1exchangeByte(0x00);

                    SSP1exchangeByte(0x00);       //trial     for
buff1

                    //Send 1 dummy byte


                    for(int z=0; z<27; z++){

                              printf("\n                    %d\t
%c",z+1,SSP1exchangeByte(0x00));

                    }

                    /*

printf("\n 1\t %c",SSP1exchangeByte(0x00));

printf("\n 2\t %c",SSP1exchangeByte(0x00));

printf("\n 3\t %c",SSP1exchangeByte(0x00));

printf("\n 4\t %c",SSP1exchangeByte(0x00));
*/

printf("\n**************************************************
**********************\n");

                    LPC_GPIO2->FIOSET = (1<<6);

          for ( i = 0; i < 10000; i++ );

                         initSSP1();
                       /*********************************/
                       printf("\nWriting from Buffer 1 to Flash
Memory\n");
                       LPC_GPIO0->FIOCLR = (1<<6);
                       //*************************Write    from
buffer1 to main memory
                //Opcode for write from buffer1 to main memory
with built-in erase
                SSP1exchangeByte(0x83);
                //Send 3 Bytes = 24 bits= 4 don't care bits +
12 addr bits (A19-A8) + 8 don't care bits
                SSP1exchangeByte(0x00);
                SSP1exchangeByte(0x00);
                SSP1exchangeByte(0x00);
               // SSP1exchangeByte(0xff);
                LPC_GPIO2->FIOSET = (1<<6);
                for ( i = 0; i < 10000; i++ );
          }

//**********************************************************
************************************************

          else if(user==2){
                    int i;
                //PART B Verification
                initSSP1();

printf("\n**************************************************
**********************\n");
                printf("\nWriting from Flash Memory to Buffer
2\n");
                LPC_GPIO0->FIOCLR = (1<<6);
                //*************************Read    from    main
memory to buff 1
                          //Opcode for Read from main memory
to buff 1
                          SSP1exchangeByte(0x55);
                          //Send 3 Bytes = 24 bits= 4 don't
care bits + 12 addr bits (A19-A8) + 8 don't care bits
                          SSP1exchangeByte(0x00);
                          SSP1exchangeByte(0x00);
                          SSP1exchangeByte(0x00);
                          LPC_GPIO2->FIOSET = (1<<6);
                          for (i = 0; i < 10000; i++ );

          printf("\n**************************************
**********************************\n");

                          initSSP1();
                          printf("\nReading    from    Buffer
2\n");
                          LPC_GPIO0->FIOCLR = (1<<6);
                          //*************************Read
from buff 1
                          //Opcode for Read buff 1

          SSP1exchangeByte(0xd6);

          //Send 3 Bytes = 24 bits= 16 don't care bits + 8
buffer addr bits ????

          SSP1exchangeByte(0x00);

          SSP1exchangeByte(0x00);

          SSP1exchangeByte(0x00);       //trial for buff1

          //Send 1 dummy byte
```

```
printf("\n",SSP1exchangeByte(0x00));
                                                for(int      z=0;
z<27; z++){

          printf("\n %d\t %c",z+1,SSP1exchangeByte(0x00));
                                                      }
                                                /*
                                                printf("\n    1\t
%c",SSP1exchangeByte(0x00));
                                                printf("\n    2\t
%c",SSP1exchangeByte(0x00));
                                                printf("\n    3\t
%c",SSP1exchangeByte(0x00));
                                                printf("\n    4\t
%c",SSP1exchangeByte(0x00));
                                                */


printf("\n*****************************************************
**********************\n");

          LPC_GPIO2->FIOSET = (1<<6);
                                                                  for
( i = 0; i < 10000; i++ );

          //LPC_GPIO0->FIOCLR = (1<<6);
          }

          }

                  while(1);
  //delayMs(0,100);
 // printf("\n1\t%x", in);
  //extern void SSPReceive( 1, uint8_t *buf, uint32_t Length );

  return 0;
}
/*************************************************************
**************
**                            End Of File
*************************************************************
*************/
```